UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/633,012	08/01/2003	Hong Wang	P15447	4733

59796 7590 01/26/2009
INTEL CORPORATION
c/o INTELLEVATE, LLC
P.O. BOX 52050
MINNEAPOLIS, MN 55402

EXAMINER	
TECKLU, ISAAC TUKU	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
01/26/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/633,012	**Applicant(s)** WANG ET AL.	
	Examiner ISAAC T. TECKLU	**Art Unit** 2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *15 October 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-4,6,7,9-31 and 33-41* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-4, 6-7, 9-31 and 33-41* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the amendment filed on 10/15/2008.

2. Claims 1-4, 6-7, 9-31 and 33-41 have been examined.

Response to Arguments

3. Applicant's arguments with respect to claims 1-4, 6-7, 9-31 and 33-41 have been considered but they are not persuasive.

The Applicant argued: "Applicant's claim 1 is explicitly preventing data of a store from replacing data to be loaded, while Hoogerrugge only describes replacing data loaded by a forwarded load with store data." (page 16).

The examiner respectfully disagrees with the above assertion. Contrary to the above assertion, Hoogerbrugge clearly teaches preventing data associated with a store instruction from being stored and preventing data from being forwarded in the following recited columns (with emphasis added):

"The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a <u>"commit" instruction is added to prevent store instructions after that location from causing a substitution with store data</u>. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program" (col.1:50-60).

"Preferably, the selective forwarding is realized by providing different versions of the load and/or store instructions. Each version defines one or more relative positions of the store and load in the pipeline. Only for instructions at those relative positions will forwarding of data from the store to the load be effected. Different versions define different sets of relative positions" (col.9:55-65).

"For example, there may be two versions of the forward load instruction. One version loads data forwarded from store instructions that start execution at N and N+1 instruction cycles after the forward load instruction. Another version loads data forwarded from a store instruction that start N instruction cycles after the forward load instruction, but not from a store instruction that starts N+1 instruction cycles after the forward load instruction" (col.9:65-67 – col.10-1-10).

"There may be different versions of the store instruction. One version forwards data to forward load instructions that start execution at N and N+1 instruction cycles before this store instruction. Another version forwards to a forward load instruction that start N instruction cycles before this store instruction, but not to a forward load instruction that starts N+1 instruction cycles before this store instruction" (col.10:5-15).

"It has been found that for some benchmark programs this allows a reduction in execution time of up to 7%. However, the reduction is frequently less than that attainable by moving load instructions past store instructions. Depending considerations of hardware cost and the applications for which the processor is intended it may therefore be preferable to include only provisions for forward load instructions and not for forward store instructions" (col:10:50-60).

"In a third step 43, once the scheduler has decided the relative scheduling of load and store instructions, it subsequently selects the required version of each load and/or store instructions (e.g. forward load in general, or forward load from store instructions N cycles later) dependent on the required dependency indicated by the second step 42 and the relative position where the load and store instructions are scheduled" (col.11:45-55).

Thus, it is respectfully submitted that the above argument is not persuasive and accordingly the rejection has been maintained as set forth in the Office Action.

Chamdani does not disclose the ability to forward speculative data or declining to forward speculative data based on whether a dependence check is successful." (page 17-18).

Examiner would like to indicate that as set forth in the Office Action, Hoogerbrugge not Chamdani discloses the ability to forward speculative data or declining to forward speculative data based on whether a dependence check is successful. Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60).

Hoogerbrugge further discloses the ability to forward speculative data or declining to forward speculative data based on whether a dependence check is successful forwarded in the following recited columns.

"Preferably, the selective forwarding is realized by providing different versions of the load and/or store instructions. Each version defines one or more relative positions of the store and load in the pipeline. Only for instructions at those relative positions will forwarding of data from the store to the load be effected. Different versions define different sets of relative positions" (col.9:55-65).

"There may be different versions of the store instruction. One version forwards data to forward load instructions that start execution at N and N+1 instruction cycles before this store instruction. Another version forwards to a forward load instruction that start N instruction cycles before this store instruction, but <u>not to a forward load instruction that starts N+1 instruction cycles before this store instruction"</u> (col.10:5-15).

"It has been found that for some benchmark programs this allows a reduction in execution time of up to 7%. However, the reduction is frequently less than that attainable by moving load instructions past store instructions. Depending considerations of hardware cost and the applications for which the processor is intended it may therefore be preferable to include <u>only provisions for forward load instructions and not for forward store instructions"</u> (col:10:50-60).

With respect to the limitation that Chamdani does not teach marking instruction information for an instruction of a speculative thread as speculative (page 5-6), examiner for the sake of argument, applies a new ground of rejection below (Kranich (US 6,574,725 B1). Thus, the arguments are moot in view of the new ground of rejection.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-4, 6-7, 9-31 and 33-41 are rejected under 35 U.S.C. 103(a) as being unpatentable over Chamdani et al. (US 2004/0073906 A1) in view of Hoogerbrugge et al. (US 6,615,333 B1) and further in view of Kranich (US 6,574,725 B1).

Per claim 1, Chamdani discloses an apparatus comprising:

a processor including: (e.g. FIG. 1, Processor 100 and related text)

preventing data associated with store instruction of the speculative thread from being forwarded to an instruction of a non-speculative thread (paragraph [0031] "… to flush a speculative thread prior to its re-execution, stores from the speculative … non speculative …").

Chamdani does not explicitly disclose blocker logic to prevent forwarding of the data associated with the store instruction to the first thread. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50).

Chamdani and Hoogerbrugge do not explicitly disclose marking logic to mark instruction information for an instruction of a speculative thread as speculative. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processros as once suggested by Kranich (col.2:10-35).

Per claim 2, Chamdani does not explicitly disclose blocker logic is further to allow the data associated with a store instruction of the speculative thread to be forwarded to an instruction of a second speculative thread. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Per claim 3, Chamdani discloses the apparatus of claim 1, further comprising: a plurality of store request buffers (e.g. FIG. 2, 230, 220 and 240 and related text), each store request buffer including a speculation identifier field (e.g. FIG. 2, 220 and related text).

Per claim 4, Chamdani discloses the apparatus of claim 1, wherein the memory system further comprises: a data cache that includes a safe-store indicator field associated with each entry of a tag array (e.g. FIG. 1, 114 and related text).

Per claim 6, Chamdani does not explicitly disclose dependence blocker logic to prevent data associated with a speculative store instruction from being forwarded to an instruction of the non-speculative thread; and store blocker logic to prevent the data from being stored in a memory system. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction <u>a commit instruction</u> is added to <u>prevent store instructions</u> after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Per claim 7, Chamdani discloses the apparatus of claim 6, wherein: store blocker logic is outside the execution pipeline (e.g. FIG. 2 and related text).

Per claim 9, Chamdani discloses the apparatus of claim 6, wherein: dependence blocker logic is included in an execution pipeline (e.g. FIG. 2 and related text).

Per claim 10, Chamdani discloses the apparatus of claim 9, wherein: dependence blocker logic is included in a memory ordering buffer of the processor (paragraph [0061] "... load/store buffer having storage locations for data and address ... buffer may comprises locations ... "and paragraph [0068] "... buffer identified by reorder buffer tags ...").

Per claim 11 (Currently Amended), Chamdani discloses a system, comprising:

a memory system that includes a memory device (e.g. FIG. 1 and related text); and

a processor associated with the memory system, the processor including dependence blocker logic to prevent data associated with a store instruction of a speculative thread from being forwarded to an instruction of a non-speculative thread (e.g. FIG. 1, Processor 100 and related text);

Chamdani discloses preventing data associated with store instruction of the speculative thread from being forwarded to an instruction of a non-speculative thread (paragraph [0031] "... to flush a speculative thread prior to its re-execution, stores from the speculative ... non speculative ..."). Chamdani does not explicitly disclose allowing the data associated with associated with the store instruction of the speculative thread to be forwarded to a speculative instruction of another speculative thread. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction <u>a commit instruction</u> is added to <u>prevent store instructions</u> after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Chamdani and Hoogerbrugge do not explicitly disclose allowing the data associated with the store instruction of the speculative thread to be forwarded to a speculative instruction of another speculative thread. However, Kranich discloses allowing the data associated with the store instruction of the speculative thread to be forwarded to a speculative instruction of another speculative thread (col.19:30-35). Therefore it would have been obvious to one skilled in the art at the time the invention was made to forward results from function unit and load/store unit and reorder buffer tag identifying the instruction being executed as once suggested by Kranich (col.2:10-35).

Per claim 12, Chamdani does not explicitly disclose the processor further includes a store blocker logic to prevent the data from being stored in the memory system. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Per claim 13, Chamdani and Hoogerbrugge do not explicitly disclose system of claim 12, wherein: the marking logic is further to associate a safe speculation domain ID with the instruction information. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processros as once suggested by Kranich (col.2:10-35).

Per claim 14, Chamdani and Hoogerbrugge do not explicitly disclose the marking logic is further to indicate a thread identifier as the speculation domain ID. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processros as once suggested by Kranich (col.2:10-35).

Per claim 15, Chamdani discloses the system of claim 12, further comprising: a store request buffer to store the speculation domain ID (paragraph [0005] "... read from global register if ... a speculative thread ..." and e.g. FIG. 2, 230 and 240 and related text).

Per claim 16, Chamdani discloses the system of claim 11, wherein: the processor includes a first logical processor to execute the non-speculative thread; and the processor includes a second logical processor to execute the speculative thread (e.g. FIG. 1, 100 and related text).

Per claim 17, Chamdani discloses the system of claim 11, further comprising: a second processor that includes said dependence blocker logic and said store blocker logic; wherein said processor is to execute the non-speculative thread and said second processor is to execute the speculative thread (e.g. FIG. 1, 112 and related text).

Per claim 18, Chamdani discloses the system of claim of claim 11, wherein: the memory system includes store blocker logic to prevent the data from being stored in the memory system and a cache organized to include a plurality of tag lines, wherein each tag line of the cache includes a unique helper thread ID field (paragraph [0005] "... read from global register if ... a speculative thread ..." and e.g. FIG. 2, 230 and 240 and related text).

Per claim 19, Chamdani discloses the system of claim 11, wherein: the memory system includes a cache organized to include a plurality of tag lines, wherein each tag line of the cache includes a safe-store indicator field (e.g. FIG. 2, 240 and related text).

Per claim 20, Chamdani discloses the system of claim 11, wherein: the memory system includes a victim tag cache to indicate evicted cache lines that include speculative load data (e.g. FIG. 2, 240 and related text).

Per claim 21, Chamdani discloses a method, comprising:

receiving instruction information for a load instruction, the instruction information including a load address (e.g. FIG. 4, S1 and related text);

performing a dependence check, wherein performing the dependence check includes:

determining if a store address of an in-flight store instruction matches the load address (e.g. FIG. 4, S2 and related text); and

Chamdani does not explicitly disclose determining if the load instruction and the in-flight store instruction each originate with a speculative thread; forwarding, if the dependence check is successful, store data associated with the in-flight store instruction to the load instruction. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Chamdani and Hoogerbrugge do not explicitly disclose declining to forward, if the dependence check is not successful, the store data to the load instruction. However, Kranich discloses declining to forward, if the dependence check is not successful, the store data to the load instruction (col.19:30-35). Therefore it would have been obvious to one skilled in the art at the time the invention was made to forward results from function unit and load/store unit and reorder buffer tag identifying the instruction being executed as once suggested by Kranich (col.2:10-35).

Per claim 22, Chamdani discloses the method of claim 21, wherein performing the dependence check further comprises: determining if the in-flight store instruction and the load instruction originate from the same thread (e.g. FIG. 4, S5 and related text).

Per claim 23, Chamdani discloses the method of claim 22, wherein determining if the in-flight store instruction and the load instruction originate from the same thread further comprises: determining if a thread ID associated with the in-flight store instruction matches a thread ID associated with the load instruction (e.g. FIG. 4, S3 and related text).

Per claim 24, Chamdani discloses the method of claim 21, wherein performing the dependence check further comprises: if the load instruction and the in-flight store instruction do not each originate with a speculative thread, determining if the load instruction and the in-flight store instruction each originate with a non-speculative thread (e.g. FIG. 4, S6 and related text).

Per claim 25, Chamdani discloses the method of claim 21, further wherein: declining to forward further comprises declining to forward the store data to the load instruction if (the load instruction and the in-flight store instruction each originate with a speculative thread) and (the in-flight store instruction originates with a speculative thread that is not older in program order than the speculative thread from which the load instruction originates) (e.g. FIG. 4, S6 and related text).

Per claim 26 (Currently Amended), Chamdani discloses a method, comprising:

determining a cache line in a cache corresponding to a speculative thread cache write request includes dirty non-speculative data (e.g. FIG.1, 110 and 4, S2 and related text)

in response to determining ~~the~~ a cache line corresponding to the a speculative thread (see at least paragraph [0014] and e.g. FIG. 1, 110 and 4, S3 and S6 and related text);

cache write request includes dirty non-speculative data (e.g. FIG. 1, element 110 and related text);

generating a write back of the dirty non-speculative data (paragraph [0038] "... currently the non-speculative ...");

Chamdani does not explicitly disclose processing a speculative thread cache write request; and processing a cache access request from a non-speculative thread. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Chamdani and Hoogerbrugge do not explicitly disclose marking the cache line as speculative. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processros as once suggested by Kranich (col.2:10-35).

Per claim 27, Chamdani discloses the method of claim 26, further comprising: forwarding speculative data from a cache to the speculative thread responsive to a data speculative thread cache read request (paragraph [0005] "... read from global register if ... a speculative thread ..." and e.g. FIG. 2, 230 and 240 and related text)

Per claim 28, Chamdani discloses the method of claim 26, comprising forwarding non-speculative store data from the ~~a~~ cache to a speculative thread responsive to a data speculative thread cache read request (paragraph [0156] "... speculative state of processor ...").

Per claim 29, Chamdani discloses the method of claim 26, further comprising forwarding non-speculative store data from the cache to a speculative thread responsive to a data speculative thread cache read request (paragraph [0038] "... currently the non-speculative ...").

Per claim 30, Chamdani discloses the method of claim 26, further comprising processing a cache access request from a non-speculative thread, wherein processing a cache access request from a non-speculative thread comprises: if a cache does not include a cache line associated with the cache access request, allocating a new cache line (e.g. FIG. 1, element 110 and related text);

wherein allocating a new cache line further comprises: if the new cache line includes dirty speculative data, allocating the new cache line without generating a write back operation (e.g. FIG. 1, element 110 and related text).

Per claim 31, Chamdani discloses the method of claim 26,further comprising allowing the speculative thread to write data to the cache if the cache line corresponding to the speculative thread cache write request includes speculative data (e.g. FIG. 1, element 110 and related text).

Per claim 33 (Currently Amended), Chamdani discloses the method of claim 26, further comprising:

if the cache does not contain data in a cache line corresponding to the speculative thread cache write request

allocating a new cache line (e.g. FIG. 1, element 110 and related text);

allowing the speculative thread to write speculative data to the new cache line (e.g. FIG. 1, element 110 and related text).

Chamdani and Hoogerbrugge do not explicitly disclose marking the new cache line as speculative. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processros as once suggested by Kranich (col.2:10-35).

Per claim 34, Chamdani discloses an apparatus comprising:

a processor including (e.g. FIG. 1, 100 and related text):

a first logical processor to execute a speculative thread (e.g. FIG. 1, 100 and related text);

a second logical processor to execute a non-speculative thread (e.g. FIG. 2, 202 and related text);

a storage area to include a speculation identifier (ID) field (e.g. FIG. 1, 102 and related text); and

Chamdani does not explicitly disclose control logic to prevent data associated with the store instruction from being consumed by the non-speculative thread, based on the speculation ID holding the first value. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Chamdani and Hoogerbrugge do not explicitly disclose the speculation ID field to hold a first value to indicate an associated store instruction is associated with the speculative thread. However, Kranich discloses the speculation ID field to hold a first value to indicate an associated store instruction is associated with the speculative thread (col.19:30-35). Therefore it would have been obvious to one skilled in the art at the time the invention was made to forward results from function unit and load/store unit and reorder buffer tag identifying the instruction being executed as once suggested by Kranich (col.2:10-35).

Per claim 35, Chamdani discloses the apparatus of claim 34, wherein: the first and the second logical processors are the same logical processor, and wherein the non-speculative thread and the speculative thread are to be time multiplexed for execution on the same logical processor (e.g. FIG. 1, 100 and related text).

Per claim 36, Chamdani discloses the apparatus of claim 34, wherein: the storage area includes a store buffer, and wherein the speculation ID field is included within a store buffer entry of the store buffer, the store buffer entry to also hold a first address associated with the store instruction and the data associated with the store instruction (e.g. FIG. 3 and related text).

Per claim 37, Chamdani discloses the apparatus of claim 36, wherein: the first value is to include a first identifier (ID) value associated with the first logical processor (paragraph [0022] "... thread will continue ... the value returned...").

Per claim 38, Chamdani discloses the apparatus of claim 37, wherein: the control logic includes comparison logic to compare a second address and a second ID value, which are associated with a load Instruction that is to be executed as part of the non-speculative thread on the second logical processor, with the first address and the first ID value (paragraph [0022] "… thread will continue ... the value returned...");

Chamdani does not explicitly disclose store blocker logic to prevent data associated with the store instruction from being consumed by the load instruction that is to be executed as part of the non-speculative thread, in response to the first ID value and the first address not matching the second ID value and the second address. However, Hoogerbrugge discloses a forward load instruction that can be taken into execution before a store instruction that stores the data loaded by the forward load instruction. The load address used by the forward load instruction is saved after it has been used to load data. Subsequently, when a store instruction is executed, the store address of the store instruction is compared with the addresses used to prefetch data into each register. If the load and store addresses address the same data, the prefetched data in the relevant register is replaced by the store data that is stored by the store instruction. The data is replaced from the time that the store instruction is completed. Thus, a register loaded with a forward load instruction always contains data that corresponds to the data that is actually in memory at the load address, no matter when the forward load instruction is executed. At the original location of the load instruction a commit instruction is added to prevent store instructions after that location from causing a substitution with store data. As a result the forward load instruction can be moved freely through the program, past any store instructions, without affecting the result of the program (col.1: 50-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to prevent the data associated with the store instruction so as to bypass the load instruction any store instruction as data will directly go to a functional unit that addresses the result as once suggested by Hoogerbrugge (col.5: 35-50, emphasis added).

Per claim 39 , Chamdani discloses the apparatus of claim 34, wherein: the processor further includes a third logical processor to execute an additional speculative thread, and wherein the control logic is to allow data associated with the store instruction from being consumed by the additional speculative thread (paragraph [0049] "… speculative thread re-executing …").

Per claim 40, Chamdani discloses the apparatus of claim 34, wherein: the first value is to include a 1-bit mode value comparison logic (paragraph [0038] "… one bit …").

Per claim 41, Chamdani and Hoogerbrugge do not explicitly disclose marking the new cache line as speculative. However, Kranich discloses marking logic to mark instruction information for an instruction of a speculative thread as speculative (col.12:35-60). Therefore it would have been obvious to one skilled in the art at the time the invention was made to mark instruction information for an instruction of a speculative thread as speculative for ease of scheduling and speculatively executing multiple threads on multiple processors as once suggested by Kranich (col.2:10-35).

Conclusion

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to ISAAC T. TECKLU whose telephone number is (571) 272-7957. The examiner can normally be reached on M-TH 9:300A - 8:00P.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Isaac T Tecklu/
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192